Box() : constructor

# Sets the following #  
val = 0

nposib = 0

posib[1-9]=[1-9]

setposib() : function to set possibilities

for( i : 1 to 9)

for( j : 1 to 9)

If sudoku[i][j]'s value is not a ZERO

.Run Loop to remove SUDOKU[i][j]'s value as a posib from corresponding row and column

.look for corresponding 3x3 box by 1-4-7 co-ordination system\*

.remove sudoku[i][j]'s value as a posib from corresponding row and column

singletons() : function to find singletons

for( i : 1 to 9)

for( j : 1 to 9)

if sudoku[i][j] has nposib==1

.find that posib and set val as that posib

# as a value has been changed, possibilities in each cell changes #

.Run Loop to remove SUDOKU[i][j]'s value as a posib from corresponding row and column

.look for corresponding 3x3 box by 1-4-7 co-ordination system\*

.remove sudoku[i][j]'s value as a posib from corresponding row and column

.goto start of the function because a value had been changed => another singleton might now be present

1-4-7 co-ordination system:

Look at value of:

i,i+1,i-1 and find whether any are 1,4,7

j,j+1,j-1 and find whether any are 1,4,7

we get pair like (1,1)(1,4)(7,4)…

this represents 3x3 matrix required

backup() : function that backup's sudoku[][]

for( i : 1 to 9)

for( j : 1 to 9)

.save[][].val = sudoku[][].val

.save[][].nposib = sudoku[][].nposib

.save[][].posib[1 to 9] = sudoku[][].posib[1 to 9]

setsudoku() : function that resets sudoku[][]

for( i : 1 to 9)

for( j : 1 to 9)

.sudoku[][].val = save[][].val

.sudoku[][].nposib = save[][].nposib

.sudoku[][].posib[1 to 9] = save[][].posib[1 to 9]

rfrequency() : function that solves based on frequency distribution

.call setposib()

.call singletons()

.counter = 0 # flag to see if changes were made or not

for( i : 1 to 9)

.freq[9]={0,0,0…0}

for( j : 1 to 9)

.if no value present in cell then add posib's values to freq[]

.n = 0

for( k : 1 to 9) # to scan through freq

if freq[k]==1

.n = k+1

.break

if(n!=0)

.counter = 1

for( j : 1 to 9)

if(sudoku[i][j] has possiblities)

if(sudoku[i][j].posib[n-1]!=0)

.set value to cell as that possibility

if(counter!=0)

.call setposib()

.call singletons()

.goto top of function

# #

# Similar code for column wise and 3x3 matrix wise solution #

# in case of 3x3 matrix, we run on following 3x3 matrices: #

# (1,1)->(1,4)->(1,7)->(4,1) ...->... (7,4)->(7,7) #

# #

guess() : function that guesses remaining solution

for( i : 1 to 9)

for( j : 1 to 9)

.first unsolved cell has its values set as the first possibility for matrix sudoku[][]

.break;

.call rfrequency()

for( i : 1 to 9)

for( j : 1 to 9)

if(sudoku is not solved yet)

if(solution is further impossible to get)

.call setsudoku() # to reset Sudoku #

.goto starting of function

# #

# if Sudoku is incorrect, program runs into infinity over here #

# #

view() : function to display final message

.print 'SUDOKU SOLVER By ABHISHEK'

# in fancy font #

INPUT OF SUDOKU IN MAIN FUNCTION

char ch;

for( i : 1 to 9)

for( j : 1 to 9)

.Get character

if(it is a valid character)

.set the value of corresponding cell of Sudoku

.display the input character

else

.goto input of character

# #

# function displays ' \* ' as guidelines for user to input #

# #